Scheme

Announcements

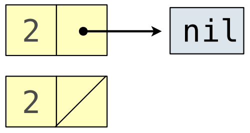
Lists

Scheme Lists

In the late 1950s, computer scientists used confusing names

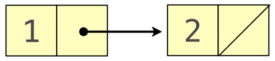
- **cons:** Two-argument procedure that creates a linked list
- **car:** Procedure that returns the first element of a list
- **cdr:** Procedure that returns the rest of a list
- **nil:** The empty list

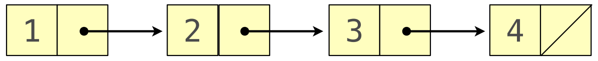
(cons 2 nil)



Important! Scheme lists are written in parentheses with elements separated by spaces

```
> (cons 1 (cons 2 nil))
(1 2)
> (define x (cons 1 (cons 2 nil))
> x
(1 2)
> (car x)
1
> (cdr x)
(2)
> (cons 1 (cons 2 (cons 3 (cons 4 nil))))
(1 2 3 4)
```





(Demo)

Symbolic Programming

Symbolic Programming

Symbols normally refer to values; how do we refer to symbols?

```
> (define a 1)
> (define b 2)
> (list a b)
(1 2)
```

No sign of "a" and "b" in the resulting value

Quotation is used to refer to symbols directly in Lisp.

```
> (list 'a 'b)
(a b)
> (list 'a b)
(a 2)
```

Short for (quote a), (quote b): Special form to indicate that the expression itself is the value.

Quotation can also be applied to combinations to form lists.

```
> '(a b c)
(a b c)
> (car '(a b c))
a
> (cdr '(a b c))
(b c)
```

(Demo)

Programs as Data

A Scheme Expression is a Scheme List

Scheme programs consist of expressions, which can be:

- Primitive expressions: 2   3.3   true   +   quotient
- Combinations: (quotient 10 2)   (not true)

The built-in Scheme list data structure (which is a linked list) can represent combinations

```
scm> (list 'quotient 10 2)
(quotient 10 2)

scm> (eval (list 'quotient 10 2))
5
```

In such a language, it is straightforward to write a program that writes a program

(Demo)

Generating Code

Quasiquotation

There are two ways to quote an expression

```
  Quote:      '(a b)   =>   (a b)

  Quasiquote: `(a b)   =>   (a b)
```

They are different because parts of a quasiquoted expression can be unquoted with ,

```
              (define b 4)

  Quote:      '(a ,(+ b 1))  =>   (a (unquote (+ b 1))

  Quasiquote: `(a ,(+ b 1))  =>   (a 5)
```

Quasiquotation is particularly convenient for generating Scheme expressions:

```
              (define (make-add-procedure n) `(lambda (d) (+ d ,n)))

              (make-add-procedure 2)  => (lambda (d) (+ d 2))
```

Example: While Statements

What's the sum of the squares of even numbers less than 10, starting with 2?

```
x = 2
total = 0
while x < 10:
    total = total + x * x
    x = x + 2
```

```
(begin
  (define (f x total)
    (if (< x 10)
      (f (+ x 2) (+ total (* x x)))
      total))
  (f 2 0)~~)~~)
```

What's the sum of the numbers whose squares are less than 50, starting with 1?

```
x = 1
total = 0
while x * x < 50:
    total = total + x
    x = x + 1
```

```
(begin
  (define (f x total)
    (if (< (* x x) 50)
      (f (+ x 1) (+ total x))
      total))
  (f 1 0)~~)~~)
```

(Demo)